Least Squares and Variations

Agnibho Roy

Fall 2020

Motivation. Least squares is a form of supervised learning that lies at the heart of prediction, using data that is given to us. This prediction can be posed as an optimization problem, and has numerous variations depending on the requirements of our problem such as specific constraints we want to satisfy, providing more weight to one data point over another, etc.

Key Idea. We first vectorize our data points into the form $Ax = y$, where A is known as the data matrix and y is a vector of our output corresponding to the data. Least squares can be boiled down to the optimization of finding the x that allows us to minimize $\|Ax - y\|_2$. Essentially, we are trying to find the "best" x in $\mathcal{R}(A)$. After producing a general solution to solve this, we try to apply it to various other optimization problems, where all we do is mold that new optimization problem in the form

$$\min \left\|A'x - y'\right\|_2$$

For some A' and y' that fits the specifications of the new optimization problem. We also provide a geometric intuition of least squares, which is helpful in understanding a lot about the properties of the least squares solution.

1 Traditional Least Squares

Let us say that we have some $y \in \mathbb{R}^m$ on some subspace $\mathcal{R}(A)$ for $A \in \mathbb{R}^{m\times n}$ of rank $r \geq 1$. We realize that the "closest" vector to y in $\mathcal{R}(A)$ is actually the orthogonal projection of y onto $\mathcal{R}(A)$, which we will call x^*. In this case, "closest" means smallest l_2 norm of the difference between Ax and y.

Theorem 1 (Least Squares)**.** The set of solutions to the optimization $\min\limits_{x\in\mathcal{R}(A)} \|Ax - y\|_2$ is

$$x^* = \{A^\dagger y + z \,:\, z \in \mathcal{N}(A)\}$$

and the one that results in the minimum norm solution is $x^* = A^\dagger y$. If A has full column rank, then from definition 10 from note 4, $A^\dagger = (A^TA)^{-1}A^T$ leading to the infamous solution $x^* = (A^TA)^{-1}A^Ty$

Proof. Let us assume that we have an orthonormal basis for $\mathcal{R}(A)$ (if not then just apply gram-schmidt). Then by the projection theorem, the projection of y onto $\mathcal{R}(A)$ can be given by:

$$\Pi_{\mathcal{R}(A)}(y) = \sum_{i=1}^{r} u_i\langle y^T, u_i\rangle = U_r(U_r^Ty) = AA^\dagger y$$

This means that our optimal x^* is just going to be the x^* in $\mathcal{R}(A)$ that gives us this projection. Mathematically,

$$Ax^* = AA^\dagger y$$
$$x^* = A^\dagger y$$

We previously introduced the concept of **affine translates** in the null space, which basically mean that if we translate our solution by any $z \in \mathcal{N}(A)$, then we still get a valid solution because $Az = 0$. This leads to the *set* of solutions $\{A^\dagger y + z \ : \ z \in \mathcal{N}(A)\}$ □

Now let us formulate the least squares as a **supervised learning** problem. Given (α_i, y_i) for $i \leq i \leq m$ (training samples) where $\alpha_i \in \mathbb{R}^n$ is the vector of features of sample i and $y_i \in \mathbb{R}$ is the response in sample i. Then we can let $A = [\alpha_1^T, \alpha_2^T \dots \alpha_n^T]^T$ and the optimization would be to find the optimal **regression coefficients** x^* that satisfies $\min\limits_{x \in \mathcal{R}(A)} \|Ax - y\|_2$ or equivalently,

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m} (\alpha_i^T x - y_i)^2$$

2 QR Decomposition method

Theorem 2 (QR Decomposition). A matrix $A \in \mathbb{R}^{m \times n}$ for $m \geq n$ and $rank(A) = n$ has a QR-decomposition, where for an orthogonal matrix $Q \in \mathbb{R}^{m \times n}$ and an upper-triangular matrix $R \in \mathbb{R}^{n \times n}$, $A = QR$

We now wish to find $x^* = [x_1, x_2, \dots, x_n]^T$, as usual. Since we have full rank, $x^* = A^\dagger y = (A^T A)^{-1})A^T y$

$$\begin{aligned} x^* &= (A^T A)^{-1} A^T y \\ &= ((QR)^T (QR))^{-1} (QR)^T y \\ &= (R^T Q^T Q R)^{-1} R^T Q^T y \\ &= (R^T R)^{-1} R^T Q^T y \\ &= R^{-1} (R^T)^{-1} R^T Q^T y \\ &= R^{-1} Q^T y \end{aligned}$$

By multiplying both sides by R^{-1}, we find that $Rx^* = Q^T y$. We inspect the matrix R closer and see that its upper triangular structure allows us to solve each element of x^* in a backwards fashion. Imagine multiplying the last row of the following matrix:

$$\begin{bmatrix} r_{0,0} & r_{0,1} & r_{0,2} & \dots & r_{0,n-1} \\ 0 & r_{1,1} & r_{1,2} & \dots & r_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & r_{n-2,n-2} & r_{n-2,n-1} \\ 0 & 0 & 0 & 0 & r_{n-1,n-1} \end{bmatrix}$$

by $x^* = [x_1 \dots x_n]^T$. Let Q_i be the ith column vector of Q, or equivalently, let Q_i^T be the ith row vector of Q^T. Then we end up getting

$$r_{n-1,n-1} x_{n-1}^* = Q_{n-1}^T y \implies x_{n-1}^* = \frac{Q_{n-1} y}{r_{n-1,n-1}}$$

Now that we know this, we can compute x_{n-2}^* and so on and so forth using the following generalized formula for solving for x_i^*:

$$r_{i,i} x_i^* + \sum_{j=i+1}^{n-1} r_{i,j} x_j^* = Q_i^T y \implies x_i^* = \frac{Q_i^T y - \sum_{j=i+1}^{n-1} r_{i,j} x_j^*}{r_{i,i}}$$

Cleverly, we never had to invert the matrix R at all. We just use this back-substitution method to iteratively get the vector x^*.

3 Least Squares Variants

So we finally have a solution to the least squares problem. Lets us now introduce some variants that still use the same solution at their hearts.

Least Squares with Equality Constraints

Let's say we want our regression coefficients x to additionally satisfy some equality constraints. For example, we want two times the first regression coefficient plus the second regression coefficient to equal 7 i.e. $2x_1 + x_2 = 7$. We can organize p such constraints into the form $Cx = d$ for some $C \in \mathbb{R}^{p\times n}$ and $d \in \mathbb{R}^p$.

Let's assume that $\tilde{x}$ is a solution to $Cx = d$ (otherwise, it is infeasible), then if $N \in \mathbb{R}^{n\times l}$ is a basis for the null space of C, then the set of *feasible* solutions to this problem is $\{\tilde{x}+Nz : z \in \mathbb{R}^l\}$ (recall again, any affine translates by vectors $z \in \mathcal{N}(C)$). Now we can revisit our least squares problem to account for *only* vectors that also satisfy our equality constraints:

$$\begin{aligned}\min_{x\in\mathbb{R}^n, Cx=d} \|Ax - y\|_2 &= \min_{z\in\mathbb{R}^l} \|A(\tilde{x} + Nz) - y\|_2 \\ &= \min_{z\in\mathbb{R}^l} \|ANz - (y - A\tilde{x})\|_2 \\ &= \min_{z\in\mathbb{R}^l} \left\|\tilde{A}z - \tilde{y}\right\|_2\end{aligned}$$

From here, the solution should follow naturally as we can just apply our general solution $z^* = \tilde{A}^\dagger\tilde{y}$ for $\tilde{A} = AN$ and $\tilde{y} = y - A\tilde{x}$. Our final answer will then be:

$$x^* = \tilde{x} + Nz^*$$

Weighted Least Squares

Sometimes we want some residuals to be more important than others because we care more if there is a deviation for a certain data point over another (ex. a certain type of customer is more important than another). In our optimization, we can then assign a weight of g_i to the residual of the data point (α_i, y_i) for $i = 1 \dots m$ to get:

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} g_i^2(y_i - \alpha_i^T x)^2 = \min_{x\in\mathbb{R}^n} \|G(Ax - y)\|_2^2$$

For some diagonal G of weights. We can actually expand G to the set of all $G \in \mathbb{S}^n_{++}$ in order to weight certain *linear combinations* of residuals, which amounts to assigning positive weights to certain *directions* in the residual space. It may be unclear *which* directions are being weighted more than another, so let us consider a visual in terms of the unit ball cost of residuals in the residual space. We can let $r = y - Ax$ for all x to be the vector of residuals, then we have:

$$\begin{aligned}\mathcal{E} &= \{Ax - y \ : \ \|G(Ax - y)\|_2^2 \le 1\} \\ &= \{r \ : \ r^T G^T G r \le 1\} \qquad (1) \\ &= \{r \ : \ r^T U_G \Lambda_G U_G^T r \le 1\} \\ &= \{U_G^T r \ : \ (U_G^T r)^T \Lambda_G (U_G^T r) \le 1\} \qquad (2)\end{aligned}$$

Recall that $A \in \mathbb{S}^n_{++} \implies A^T A \in \mathbb{S}^n_{++}$ and we can use the spectral decomposition of $G^T G$. Now we can plot the unit ball as defined in (1) as an ellipsoid whose axes are aligned to the eigenvectors u_i of $G^T G$. The ellipsoid explains the following concepts about the weighting:

1. The eigenvector corresponding to the semi-axis length $\lambda_{\min}^{-1}$ is the largest one, which means that the residual vectors can be super large but will still be bounded by a cost of 1. The weight is assigned lowest in this direction (less sensitive).
2. The eigenvector corresponding to the semi-axis length $\lambda_{\max}^{-1}$ is the smallest one, which means that the if the residual vectors increase by just a little bit, then they exceed a cost of 1. The weight is assigned the highest in this direction (more sensitive).
3. Combining the above two points, this means that $\lambda_1 > \lambda_2 \dots \lambda_m$ act as the residual weights corresponding to directions $u_1 \dots u_m$. The norm ball defined as (2) just provides the ellipsoid rotated to be aligned with the standard x and y axes.

Total Least Squares

We can actually reformulate our traditional least squares problem into a **perturbation** problem given by

$$\min_{\tilde{y}\in\mathbb{R}^m, y+\tilde{y}\in\mathcal{R}(A)} \|\tilde{y}\|_2^2$$

In other words, we are trying to induce a small change to our response y by $\tilde{y}$ so that $y+\tilde{y}$ lies in the range space of A. Out of all such perturbations that lies in $\mathcal{R}(A)$, we are seeking one of *least* cost.

Total least squares takes this perturbation a step further, and gives us an extra degree of freedom by allowing us to perturb matrix A by some $C \in \mathbb{R}^{m\times n}$. In other words, we want to seek the **least cost perturbations** $C \in \mathbb{R}^{m\times n}$ and $\tilde{y} \in \mathbb{R}^n$ such that the *perturbed* response vector lies in the *perturbed* range space - $y+\tilde{y} \in \mathcal{R}(A+C)$. We measure the cost in the Frobenius norm sense, summarized below:

$$\min_{C\in\mathbb{R}^{m\times n}, \tilde{y}\in\mathbb{R}^m, y+\tilde{y}\in\mathcal{R}(A+C)} \left\| \begin{bmatrix} C & \tilde{y} \end{bmatrix} \right\|_F$$

Sometimes this optimization may not have a solution actually. For example, consider the following perturbation of A and y as defined below:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} C & \tilde{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \epsilon & 0 \end{bmatrix}$$

Clearly, any $\epsilon > 0$ will work because it will make y come into the range space of $A+C$, since it now goes from spanning only $\mathbb{R}^1$ to $\mathbb{R}^2$. However 0 itself itself does not work, so there is no defined solution.

4 Polynomial Regression

Least squares attempts to build a linear predictor, but we want to expand this to get an affine prediction (meaning that we also explain linear models that may not pass through the origin) and nonlinear mappings. For an affine translate we can simply prepend the term 1 to each of our α_i features to simulate some sort of b term for $\hat{y} = b + \alpha^T x$. We can also simulate nonlinear mappings to a polynomial by taking each data point and taking it to the power d to represent training the coefficient for some x^d in our model. So lets define some translation of our data points (α_i, y_i) as the following:

$$\phi(\alpha) = \begin{bmatrix} 1 & \alpha & \dots & \alpha^{d-1} \end{bmatrix}^T$$

$$\Phi = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \phi(\alpha_1)^T & \phi(\alpha_2)^T & \dots & \phi(\alpha_m)^T \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{d\times m}$$

Now we can define the optimization:

$$\min_{x\in\mathbb{R}^d}\sum_{i=1}^{m}\left(\phi(\alpha_i)^T-y_i\right)^2=\min_{x\in\mathbb{R}^d}\left\|\Phi^T x-y\right\|_2^2$$

From here, the solution is going to be $x^*=\Phi^{T\dagger}y$. We can expand polynomial regression to not only be powers of a single feature, but of multiple features as well. For example, the predictor could be something of the form $\hat{z}=b+w_1x+w_2x^2+w_3y+w_4y^2$.

How do we actually pick the value of d? The answer is cross validation. We restrict the training sample to a fraction of the overall (say 80%), then run a supervised learning algorithm on remaining (20%) training data. Finally, we test it on the remaining samples (**validation samples**), and repeat with randomly chosen subsets of training samples. We assess the test set based on some error function, and find the value of d that reduces this error. Generally, when our model complexity increases, our validation error first decreases then increases again.

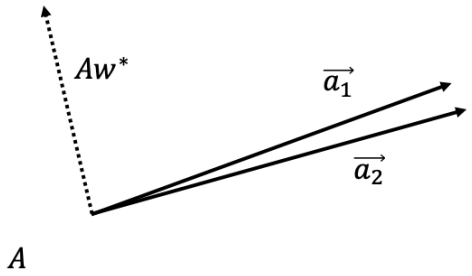
5 Regularization

There are times when the data lends to coefficients taking on large and often extreme values, and is due to us being *too aggressive* in an attempt to fit all the data. One common example is when there is "bad data" of some sort, where small values in the data matrix lead to large coefficients. For example, consider the following data matrix and y value:

$$A=\begin{bmatrix}1 & 0\\ 0 & \epsilon\end{bmatrix}=\begin{bmatrix}\vec{a}_1 & \vec{a}_2\end{bmatrix},\quad y=\begin{bmatrix}1\\1\end{bmatrix}$$

It should be clear that we can get y to be in the range space of A because they are of the same dimension, and specifically taking the linear combination $\vec{a}_1+\frac{1}{\epsilon}\vec{a}_2$ will lead to this. Now, for small ϵ, this leads to very high coefficients.

Another very common example is when we have multilicollinearity in A. As an intuitive way to explain why this is a problem, consider two "nearly collinear" vectors visualized spanning some subspace A in two dimensions. Moreover, let Ax^* represent the projection of the vector y onto (the optimal solution to the traditional least squares).



If we want to represent Ax^* (which has a very low inner product with each of $\vec{a}_1$ and $\vec{a}_2$) as a linear combination of a_1 and a_2, then to *spread* the angle between a_1 and a_2 to *reach* Ax^* is going to make the coefficients very large. Try this with a small example. In order to fix it, we do do the following:

Theorem 3 (Ridge Regression)**.** We pick a hyperparameter $\lambda>0$ that solves:

$$\min_{x\in\mathbb{R}^d}\|Ax-y\|_2^2+\lambda\|x\|_2^2$$

Which amounts to penalizing our coefficients to be too small or too large in some way. If $rank(A) = n$, then the solution is:

$$x^* = (A^T A + \lambda I_n)^{-1} A^T y$$

Proof. We can rewrite our minimization problem as:

$$\min_{x \in \mathbb{R}^d} \left\| \begin{bmatrix} A \\ \sqrt{\lambda} I_n \end{bmatrix} x - \begin{bmatrix} I_m \\ 0^{n \times m} y \end{bmatrix} \right\|$$

Which lends itself to the solution:

$$\begin{bmatrix} A \\ \sqrt{\lambda} I_n \end{bmatrix}^\dagger \begin{bmatrix} I_m \\ 0^{n \times m} y \end{bmatrix} = (A^T A + \lambda I_n)^{-1} A^T y$$

□

Another form of regularization, which is actually a weighted version of ridge regression, is known as Tikhanov regularization. It is defined as:

Theorem 4 (Tikhanov Regularization)**.** For $G_1 \in \mathbb{S}^m_{++}, G_2 \in \mathbb{S}^n_{++}$, and $x_0 \in \mathbb{R}^n$, the minimization objective is:

$$\min_{x \in \mathbb{R}^d} \|G_1(Ax - y)\|_2^2 + \lambda \|G_2(x - x_0)\|_2^2$$

Essentially, we believe a priori that the vector of regression coefficients should be close to x_0 and as a result, penalize deviations from this prior belief on x via G_2. We penalize our vector of residuals via G_1 as we do with normal weighted least squares. Here is the final regularization method that we will discuss:

Theorem 5 (LASSO)**.** The minimization objective is:

$$\min_{x \in \mathbb{R}^d} \|Ax - y\|_2^2 + \lambda \|w\|_1$$

Over time, LASSO (least absolute shrinkage and selection operator) became widely popular because it encourages sparsity in the optimal x. Lots of real world problems appear to admit sparse models and sparsity is good for the following reasons:

1. Easier to store the vector of regression coefficients
2. Easier to transmit the vector of regression coefficients
3. Easier to compute response to a new feature vector